

This project will help you gain hands-on experience with automation, containerization, Kubernetes, monitoring, and security.

Tech Stack

- **Version Control:** Git, GitHub/GitLab
 - **CI/CD:** Jenkins, GitHub Actions, or GitLab CI/CD
 - **Containerization:** Docker
 - **Orchestration:** Kubernetes (Minikube, K3s, or EKS/AKS/GKE)
 - **Configuration Management:** Ansible
 - **Infrastructure as Code (IaC):** Terraform
 - **Monitoring & Logging:** Prometheus, Grafana, ELK Stack
 - **Security:** SonarQube, Trivy, HashiCorp Vault
-

Step-by-Step Implementation Guide

1. Setup & Code Repository

- Install Git and create a GitHub repository for a simple microservices-based application (Node.js, Python, or Java).
- Clone the repository to your local machine and set up a basic project structure.
- Follow a **Git branching strategy** (main, dev, feature branches) for better code management.

2. Dockerize the Application

- Install Docker on your system.
- Write a **Dockerfile** for each microservice to define how it should be containerized.
- Create a **docker-compose.yml** file for local testing.
- Build and run the containers using Docker Compose.
- Push the Docker images to **Docker Hub** or a private registry.

3. CI/CD Pipeline Implementation

- Install Jenkins or set up GitHub Actions.
- Configure a **Jenkins pipeline** or **GitHub Actions workflow** to:
 - **Trigger builds** on code push.
 - **Run tests** (unit & integration).
 - **Build and push Docker images** to Docker Hub or AWS ECR.
 - **Deploy to Kubernetes automatically.**

4. Deploy Application to Kubernetes

- Install Kubernetes locally (Minikube) or use a managed service (EKS/GKE/AKS).
- Write **Kubernetes manifests** (Deployments, Services, Ingress).
- Use **Helm charts** to manage deployments.
- Apply the Kubernetes manifests using `kubectl apply -f`.
- Verify the deployment using `kubectl get pods, services`.

5. Infrastructure Automation

- Install **Terraform** and write Terraform scripts to provision cloud infrastructure.
- Deploy Kubernetes cluster on AWS/GCP using Terraform.
- Use **Ansible** for automating server configurations and Kubernetes node setup.

6. Monitoring & Logging

- Install **Prometheus & Grafana** for real-time metrics monitoring.
- Deploy **ELK Stack (Elasticsearch, Logstash, Kibana)** for centralized logging.
- Configure alerts in Grafana for key application metrics.

7. Security & Compliance

- Scan Docker images with **Trivy** to detect vulnerabilities.
- Use **SonarQube** for static code analysis and quality checks.
- Implement **HashiCorp Vault** for secure storage of secrets.
- Enforce security best practices in the CI/CD pipeline.

8. Auto-Scaling & Load Balancing

- Configure **Horizontal Pod Autoscaler (HPA)** in Kubernetes for scaling based on CPU/memory usage.
- Use **Ingress Controller (NGINX or Traefik)** for load balancing and routing traffic.
- Set up **Kubernetes Cluster Autoscaler** to automatically adjust node count.

Expected Outcome

- Fully automated CI/CD pipeline.
- Kubernetes-based microservices deployment.

- Automated monitoring and security scanning.
- Scalable and resilient application.

By completing this project, you will gain real-world DevOps experience that you can showcase on your resume. ☐ Let me know if you need any additional details!